COMPUTING CURVATURE FOR VOLUME OF FLUID METHODS USING MACHINE LEARNING

by
Yinghe Qi

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science in Engineering

Baltimore, Maryland
May, 2021

Abstract

In spite of considerable progress, computing curvature in Volume of Fluid (VOF) methods continues to be a challenge. The goal is to develop a function or a subroutine that returns the curvature in computational cells containing an interface separating two immiscible fluids, given the volume fraction in the cell and the adjacent cells. Currently, the most accurate approach is to fit a curve (2D), or a surface (3D), matching the volume fractions and finding the curvature by differentiation. Here, a different approach is examined. A synthetic data set, relating curvature to volume fractions, is generated using well- defined shapes where the curvature and volume fractions are easily found and then machine learning is used to fit the data (training). The resulting function is used to find the curvature for shapes not used for the training and implemented into a code to track moving interfaces. The results suggest that using machine learning to generate the relationship is a viable approach that results in reasonably accurate predictions.

Primary Reader and Advisor: Gretar Tryggvason

Secondary Reader: Jiacai Lu

Acknowledgements

I am very grateful to my advisor, Professor Gretar Tryggvason, for introducing me to the amazing world of computational fluid dynamics and machine learning, and for his guidance along the road of my master's study. Prof. Tryggvason's profound insights and broad vision in this area have deeply inspired me and have been a role model to me. I would also like to thank him for his patient, support, and encouragement he gave me. Without his guidance and feedback, this thesis would not have been achievable.

I would also like to express my special acknowledgments to Prof. Jiacai Lu for the helpful discussion and assistantship. I am also grateful to Professor Rui Ni, who is also my Ph.D. advisor, for always being supportive when preparing this thesis. My deep appreciation also goes out to all my research team members. Their suggestions and support make the research process enjoyable.

The project presented in this thesis has been published in *Journal of Computational Physics* (Qi, Yinghe, et al. "Computing curvature for volume of fluid methods using machine learning." *Journal of Computational Physics* 377 (2019): 155-161.).

Contents

List of Figures

Chapter 1

Introduction

Computing the evolution of a fluid interface separating immiscible fluids goes back to the beginning of CFD ([1]). The most common approach is to use one grid for the whole domain and solve the governing equations simultaneously for both fluids using the so-called one-fluid formulation where the different fluids are treated as one fluid with different material properties and a singular source term is added to account for surface forces. The use of the one-fluid form of the governing equations requires an index or marker function to identify the different fluid and several methods are available to advect the index function ([2]). Of those, the Volume of Fluid (VOF) approach, where volume fraction of one fluid is used as an index function, is one the oldest ([3]) and most widely used strategy.

One of the biggest challenge in VOF methods is the computation of surfaces forces. In early versions including surface tensions was very difficult, but major progress was made by the introduction of the Continuous Force ([4]) method which made it possible to compute the surface force at fluid interfaces in a reasonably reliable way, by numerically differentiating an normal vector field extended off the interface and using the geometrical identity $\kappa = \nabla \cdot \mathbf{n}$. A similar strategy was also used by [5] who worked directly with the stress field. Those approaches, coupled with the Piecewise Linear Interface Calculation strategy ([6] to advect the volume fraction, played a major role in making the VOF method one of the most—if not the most—used method for

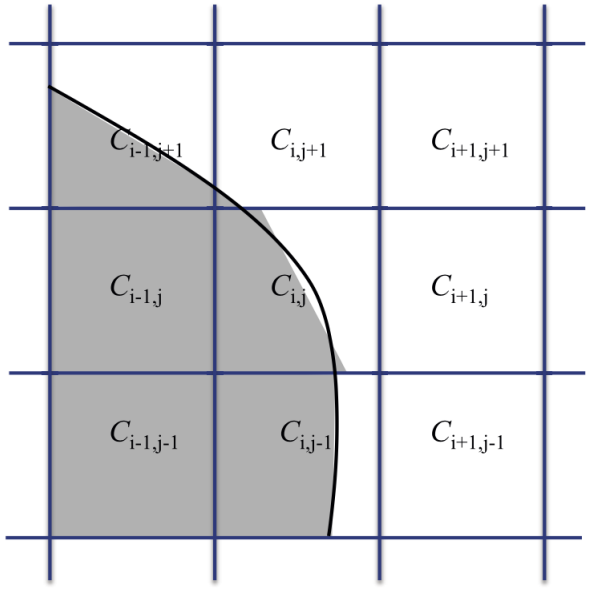

Figure 1-1. Sketch showing an interface crossing several computational cells and the volume fractions constructed by the PLIC method.

simulations of flows with sharp, moving interfaces. More recent methods have improved on the above technique by using 1) So called "balanced-force methods" (defined in [7] but also used previously by [8]) and 2) so-called Height Functions that allow to find a point on the interface with fourth order accuracy ([9]). The Height-Function method yields precise results in 2D ([10]) allowing to decrease spurious currents to machine accuracy and somewhat less precise results in 3D ([11]). Despite this success, the Height-Function method is limited to situations where the number of grid points per radius of curvature is about ten or more. For smaller radii, one needs to resort to various kinds of fitting. One either directly fits the area or volume under a curve to the volume fraction ([8]) or finds approximate points on the interface and fits a paraboloid curve or surface the set of points. More complex methods have been suggested ([11]). In all cases the accuracy is low and the error can be of order one for a few points (less than five or ten) in the radius of curvature. A review of recent numerical methods for surface tension may be found in [12].

Figure 1-1 shows an interface cutting though several cells and the void fraction in

each cell. Although we show the original interface in the figure, in actual computations the only information available is the volume fraction in the cell that we are considering and the adjacent cells. We focus on two-dimensional flow below, to simplify the discussion, but the approach should carry over to fully 3D flows in a straightforward way. To find the curvature in each interface cell, $\kappa_{i,j}$, we seek a functional relationship

$$h\kappa_{i,j} = f\left(\begin{bmatrix} C(i-1,j+1) & C(i,j+1) & C(i+1,j+1) \\ C(i-1,j) & C(i,j) & C(i+1,j) \\ C(i-1,j-1) & C(i,j-1) & C(i+1,j-1) \end{bmatrix}\right) \tag{1.1}$$

relating the curvature to the nine volume fractions in and around the cell denoted by (i, j). Since the curvature has dimension one over length, it is multiplied by the cell width h (assuming that the cell height and width are the same), to make the expression nondimensional.

Here we examine an alternative approach and find the relation between curvature and the volume fractions using machine learning. The closest approach to the one described here was developed by [13], where the curvature is found by fitting a database generated using circles of different sizes. However, instead of using the nine volume fractions in equation (1.1), three variables accounting for the volume fraction in the nine cells; the volume fraction in the center cell; and the "tilt" of the interface are used.

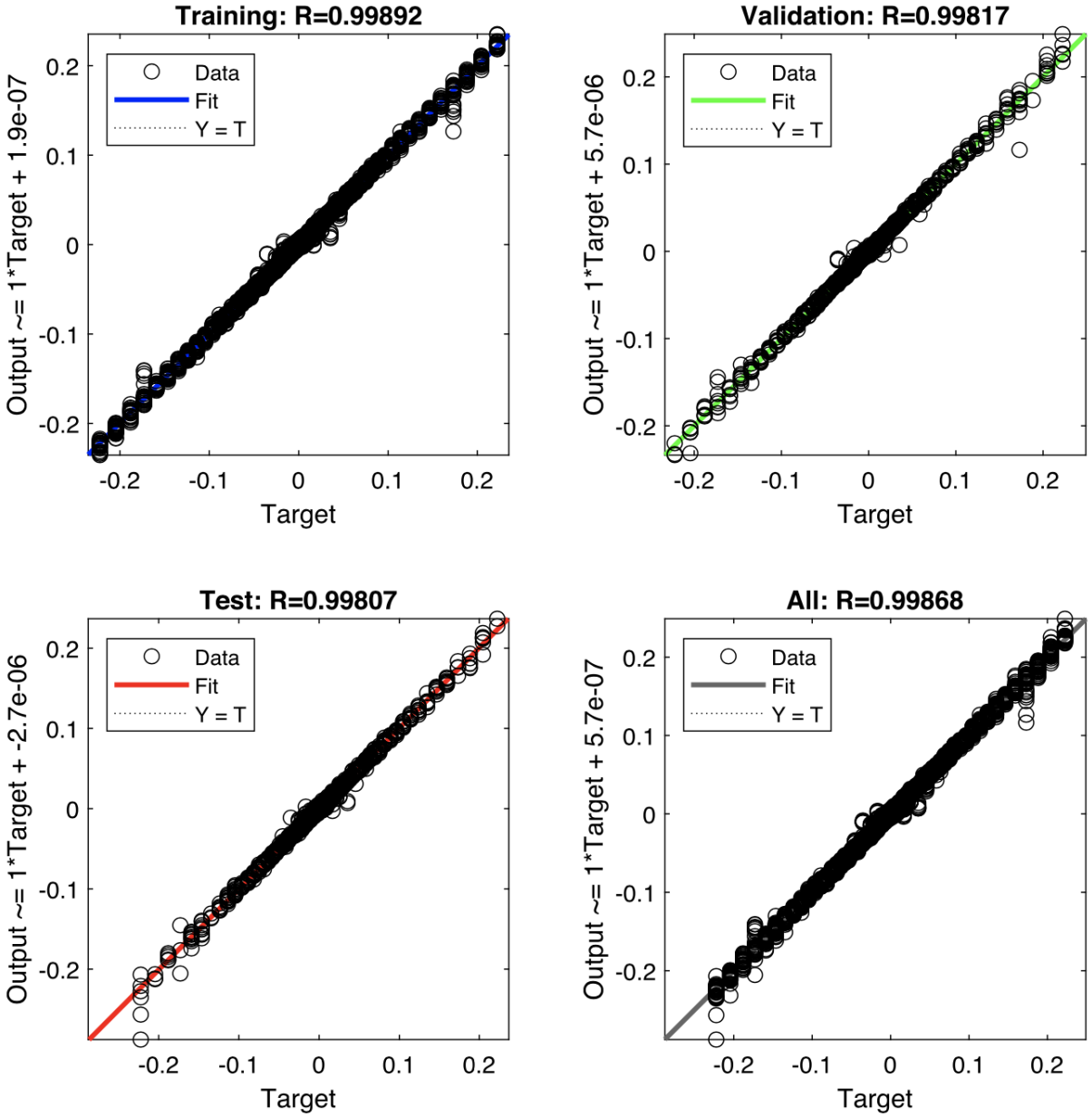

Figure 1-2. Plots of the fitted curvature versus the exact curvature. (a) training data; (b) test data; (c) validation data; (d) complete dataset. Plots generated by the Matlab Neural Network Toolbox.

Chapter 2

Learning the curvature and testing

Finding the functional relationship between curvature and volume fractions—as expressed by equation (1.1)—using Machine Learning, involves three steps:

- Generation of a synthetic dataset, spanning a large range of interface curvatures and orientations with respect to the grid.
- Fitting the data using Neural Networks (the learning stage) to find the curvature as a function of the volume fractions.
- Testing the fit using shapes not used for the fitting

To generate a data set containing curvature and volume fractions for well-defined shapes where the curvature and volume fractions are easily found, we use circles of varying sizes (as in [13]). The circles are placed on a grid, and the volume fraction in each cell found. Since we work with the curvature scaled by the grid size, one grid is sufficient. The volume fraction in each grid cell is found by integrating the area under a circle crossing the cell and for cells away from the interface the volume fraction is either zero or one and can be determined by checking if the distance from the center of the cell to the center of the circle is shorter or longer than the radius of the circle. For each circle we gather data for both positive and negative curvatures, the sign defined by whether the circle or the outside fluid is filled with the marker identified as

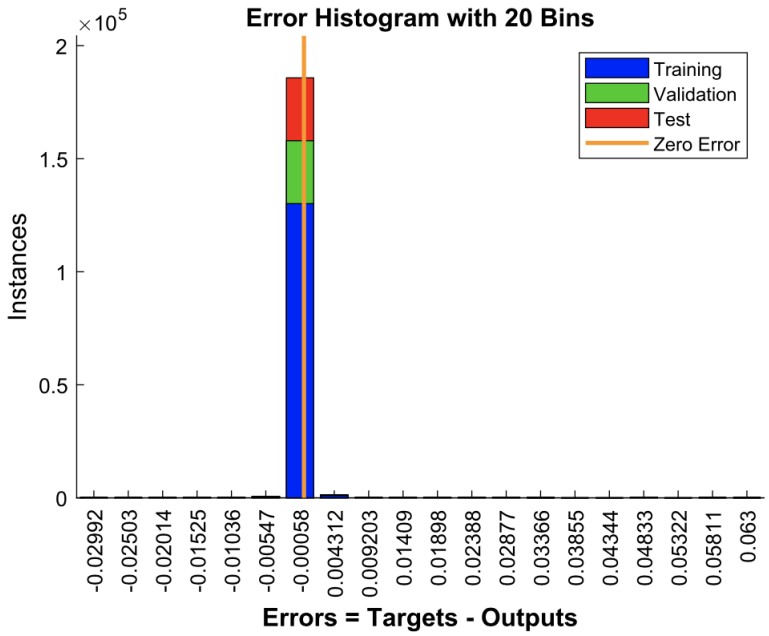

Figure 2-1. The distribution of the error for the fit shown in figure 2. Plots generated by the Matlab Neural Network Toolbox.

$C = 1$. By using circles we first of all ensure that the single curvature for each circle is known exactly and we do not have to deal with the question of exactly where in a cell it is computed, and secondly, our data is completely symmetric with respect to rotations and reflections. For the results presented here our data set is generated using a 1×1 domain resolved by a fixed grid of 2000×2000 grid points and circles of different radii with a center in the middle of the grid. The radius of the circles ranges from 0.00225 to 0.475 (so that there are 9 grid points across the smallest circle). The scaled curvature ($h\kappa$ thus ranges from 0.001 to 0.222. A total of 65 circles are used, distributed using an exponential function to sample the small radii (high curvature) region more densely. This results in 187,976 rows of data, each relating the curvature and the nine void fractions. We have also used different distributions of the radii (such as piecewise linear) and generally find similar results, as long as the whole range is covered.

The fitting is done using Matlab and the build in Neural Network Toolbox. The

curvatures constitute a $1 \times NumData$ array of "targets" and the void fractions a $9 \times NumData$ array of inputs, where $NumData$ is the total number of relationships (187,976 for the dataset used here). After the data has been loaded, the Matlab function *train* is used to relate the curvature to the void fractions. We use the default settings of a two-layer feed-forward network with 10 sigmoid hidden neurons and linear output neurons using Levenberg-Marquardt process for the back propagation.The Matlab package that we are using does not allow us to change the number of layers but we have experimented with different number of neurons and found that while the results get worse for much fewer neurons, only slight improvement is seen by increasing the number beyond that used here. The data is divided in three parts for training, testing and validation, and we use the default of 70%, 15% and 15%, respectively. The training stops when the mean square error of the validation sample stops improving or the maximum number of steps (1000 in our case) is reached. At that point the network is saved as a Matrix-Only Function. Figure 1-2 shows the quality of the fit. The curvature as found by the neural network is plotted versus the exact curvature for each point in the data set. If the fit was perfect, all the points should lie on the 45 degree line. In (a) fit is shown for the training data; (b) shows the same thing for the test data; In (c) we examine the validation data; and in (d) the fitted curvature and the exact curvatures are compared for the complete dataset. The solid line is a least squares fit through the points shown in each frame. Figure 2-1 shows the error distribution for the final fit. Obviously, most of the training data points have close to zero error, but even the validation points are closely distributed around the origin.

The fitting (or learning) results in a function that is saved and can be called from other programs to find the curvature for values of the volume fractions resulting from arbitrarily shaped interfaces. Fig. 2-3 shows the structure of the network, as reported by Matlab. For a two-layer network the outputs, y_k, are related to the inputs, x_i,

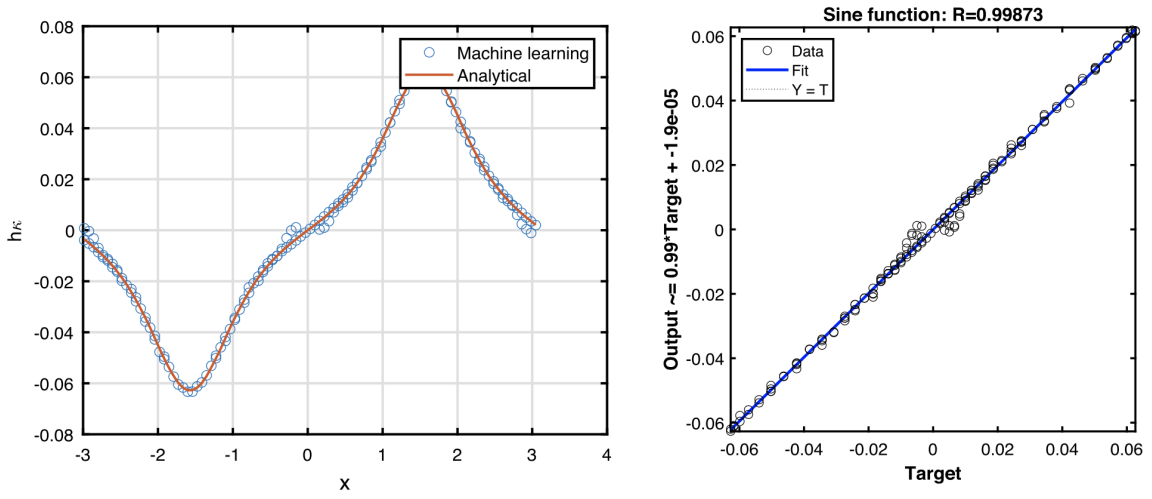

Figure 2-2. (a) Plots of the curvature found by machine learning (circles) and the exact curvature (line) for a sine wave. (b) The curvature found by machine learning versus the exact curvature.

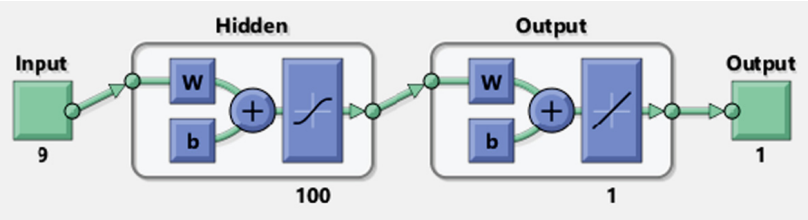

Figure 2-3. A schematic of the Neural Network, showing the nine inputs (volume fractions), the 100 hidden neurons and the single output (the curvature). Here, **w** denotes the weights and **b** is the bias vector.

through the general expression:

$$y_k = g_2\left(\sum_{j=1}^{N_N} w_{kj}^2 g_1\left(\sum_{i=1}^{N_i} w_{ji}^1 x_i + b_j^1\right) + b_k^2\right). \tag{2.1}$$

Here, the g's are the activation functions, N_N is the number of neurons and N_i the number of inputs. In our case there is only one output so $k = 1$ and $y_1 = h_\kappa$, x_j are the nine volume fractions so $N_i = 9$, and for hundred neurons in the hidden layer $N_N = 100$. The weights w_{lm} and the biases b_l are adjusted during the "learning."

To test the fit on a more complex curve, where the curvature varies along the interface, we look at a large amplitude sine wave, defined by $y(x) = A\sin(x)$. The curvature can be found analytically: $\kappa(x) = -A\sin(x) \times (1 + A^2\cos^2(x))^{-3/2}$. For

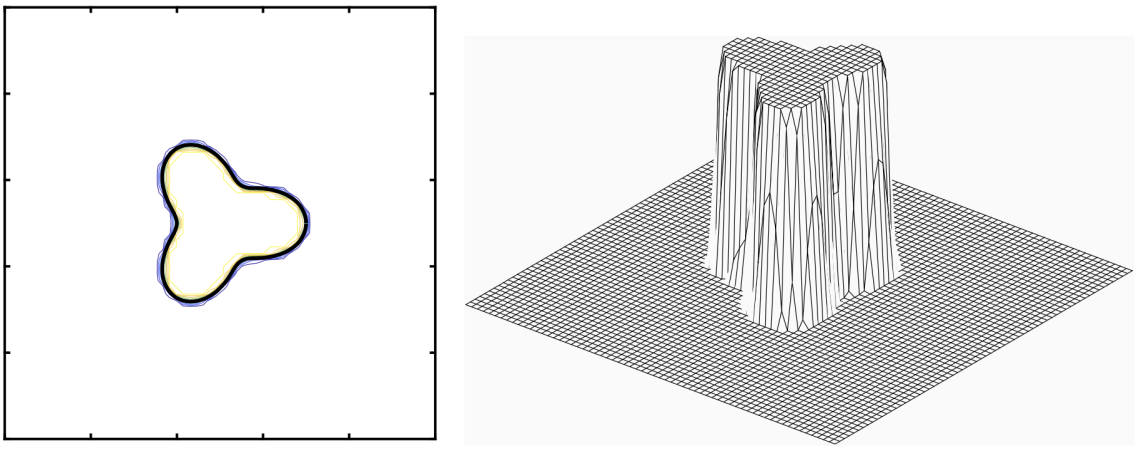

Figure 2-4. The initial conditions used to test the curvature routine generated by the neural network.

the results presented here $A = 1.0$. The curve divides a rectangular $2\pi \times 2\pi$ domain into two regions, and the value of the volume fraction of an index function is set to unity in cells below the curve and zero above the curve. The domain is resolved by 100×100 grid and the volume fraction in cells crossed by the curve is constructed in the same way as for the learning cases, by representing the curve by connected marker points and finding the fractional area on one side of the curve. The curvature was calculated analytically for the midpoint of each cell. In figure 2-2(a) we plot the fitted curvature and the exact curvature versus the x coordinate, for one value of the amplitude. Overall the agreement is reasonably good, although the points near zero curvature show some scatter. The quality of the fit is also shown in figure 2-2(b), where the curvature from the neural network is plotted versus the exact curvature. If the agreement was perfect, all the points should lie on the 45-degree line. We have also tested the fit using larger amplitudes for the sine function and different grid resolution and find similar results. As an aside we mention that initially our data set lacked points for very small curvatures (nearly flat interfaces) and in those cases we generally saw large errors where the curvature was small. A more complete data set solved this problem, emphasizing the need for the data to span all possible cases.

Chapter 3

Implementation in a Flow Solver

To test how finding the curvature by machine learning works in a flow solver, we have implemented the function developed by the neural network machine learning in a finite-volume/front-tracking code, where the interface is tracked by connected marker particles advected by the flow. The flow solver is a standard finite volume scheme implemented on a regular staggered grid, where all spatial variables are approximated by centered second-order finite differences and the time integration is done by a third-order Runga-Kutta method ([2]). The interface is represented by connected marker points and the volume fraction (and thus the index function) is found directly from the front ([14, 15]), in the same way as for the learning data and the sine function example above, and used to find the curvature. The interface is smoothed slightly at each time step to suppress small "wiggles." The surface tension is computed in each pressure cell and averaged to find the value at the velocity nodes. To find the surface force we follow a procedure similar to [13] and first find the normal to the interface by differentiating the volume fraction field, and then use it and the volume fraction to find a straight line approximation, Δl, to the length of the interface in each cell. The surface force per unit volume is then given by $\mathbf{f}^{\sigma}_{i,j} = \sigma\kappa_{i,j}\mathbf{n}_{i,j}\Delta l_{i,j}/h^2$. The surface tension is computed in each pressure cell and averaged to find the value at the velocity nodes. We note that for interface cells we divide the surface force by the average density of the different fluids, following [13], instead of the actual density in the cell.

The interface is smoothed slightly at each time step to suppress small "wiggles."

Figure 2-4 shows the initial conditions for one case. Here the initial interface is a three-armed star defined by $r(\theta) = R_o + A_o \times \cos(n\theta)$ with $n = 3$, $R_o = 0.15$, and $A_o = 0.05$, defining a drop with density $\rho_d = 2.0$ and viscosity $\mu_d = 0.02$. The ambient fluid has $\rho_o = 1.0$, viscosity $\mu_o = 0.01$ and surface tension $\sigma = 10$. We follow the motion of the drop as surface tension pulls it into a circular shape. In figure 2-4 the interface and contours of the index function are shown on the left and a 3D view of the index function on the left. The fluid solver uses a 64^2 grid to resolved the 1×1 domain, a time step equal to $\Delta t = 0.005$, and a SOR iteration to solve the pressure equation. All boundary conditions are taken to be no-slip walls.

The interface shape, contours of the index function and the velocity field at one time are shown in figure 3-1 for two different grid resolutions. At the time the shape been "inverted" in the sense that "valleys" have replaced "bulges" and vice versa, although the amplitude is still growing, as seen in the velocity field.

A more quantitative comparison is shown if figure 3-2a where the moment of the drop is plotted versus time. The moment is defined by

$$M(t) = \int\int C(x,y)\Big((x-x_0)^2 + (y-y_0)^2\Big)dxdy \tag{3.1}$$

where (x_0, y_0) are the center of the drop. The moment is found using the same fitting but three different grid resolutions and it is clear that the results on the two finest grids are relatively close. In machine learning the fit is not an exact relationship so repeated fitting does not, in general, result in identical results. To test the sensitivity of the results to the specific fit we show the moment versus time in figure 3-2b as found by three different fits. Although a very slight difference can be detected near the end, for most of the time shown all three fits result in essentially identical results.

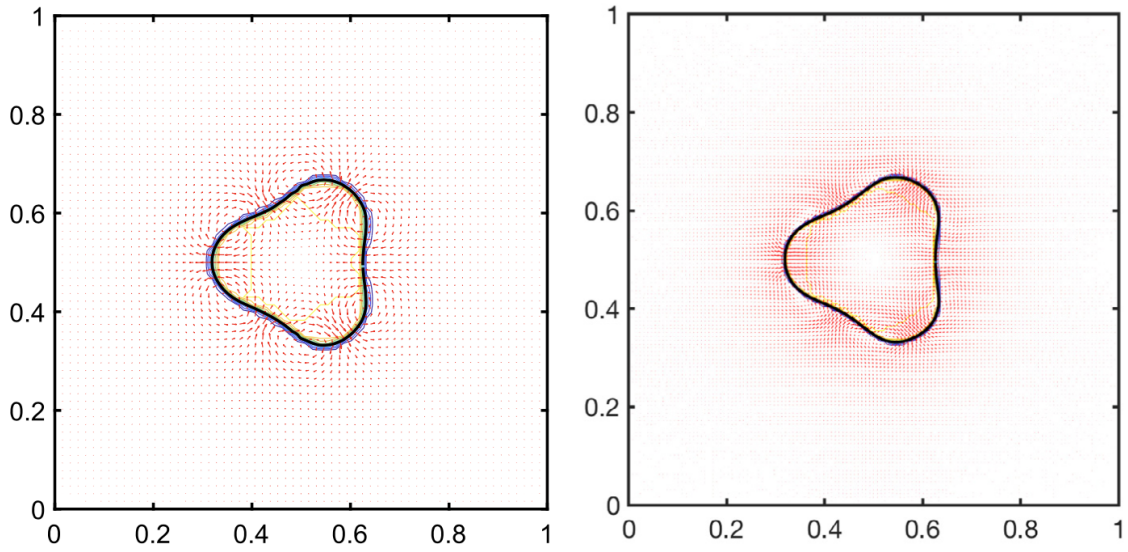

Figure 3-1. The interface shape and the velocity field at time 0.02 as computed on a 64^2 grid (left) and a 128^2 grid (right).

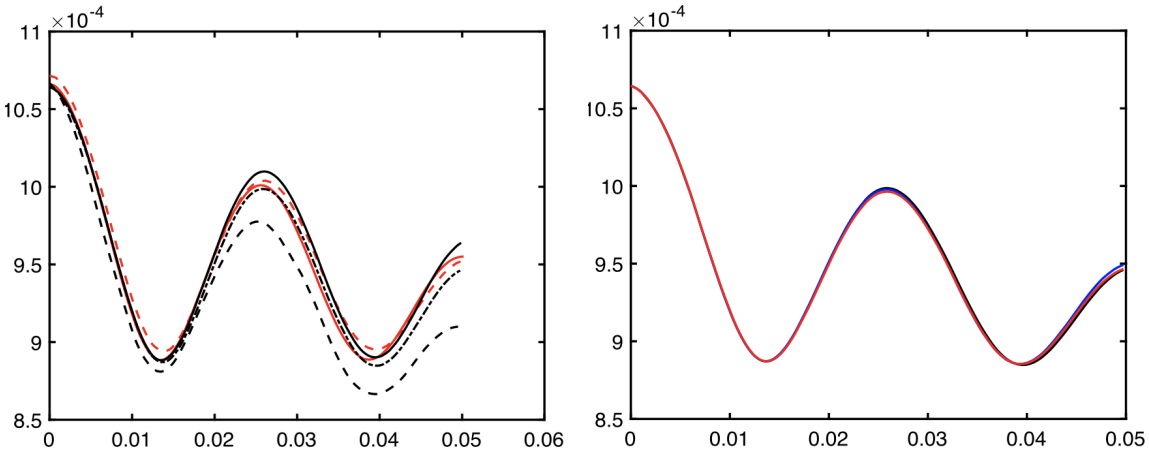

Figure 3-2. The moment of the drop versus time. (a) Results computed on a 64^2 grid (black), a 128^2 grid (blue) and 256^2 grid (red). (b) Results using three different fits computed on a 128^2 grid.

Conclusions and general discussion

Finding a relationship between the volume fractions and the curvature in VOF methods has been a topic of considerable interest since the pioneering paper by [4]. Here we have showed that using machine learning to extract the relationship from an dataset, generated using circles of variable sizes, is able to capture the curvature of a more complex interface with a spatially varying curvature reasonably well. The results suggest that when it is complicated to generate an exact relationships from fundamental considerations, but easy to generate a synthetic dataset from simple examples, this approach may be a viable alternative to more conventional derivations. For the application described here we could keep the learning cases extremely simple, but more complex cases are easily generated. We note that we have not tested the efficiency of this method compared to more traditional ones, and no attempt has been made to make the function efficient. It is is likely that there are significant opportunities to do so. We also note that while the accuracy is easily established for a given shape and numerical parameters, there is no explicit order that guarantees convergence under grid refinement.

References

1. Harlow, F. H. & Welch, J. E. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. en. *Physics of Fluids* **8,** 2182 (1965).
2. Tryggvason, G., Scardovelli, R. & Zaleski, S. *Direct numerical simulations of gas-liquid multiphase flows* OCLC: ocn664324526 (Cambridge University Press, Cambridge ; New York, 2011).
3. Hirt, C. & Nichols, B. Volume of fluid (VOF) method for the dynamics of free boundaries. en. *Journal of Computational Physics* **39,** 201–225 (Jan. 1981).
4. Brackbill, J., Kothe, D. & Zemach, C. A continuum method for modeling surface tension. en. *Journal of Computational Physics* **100,** 335–354 (June 1992).
5. Lafaurie, B., Nardone, C., Scardovelli, R., Zaleski, S. & Zanetti, G. Modelling Merging and Fragmentation in Multiphase Flows with SURFER. en. *Journal of Computational Physics* **113,** 134–147 (July 1994).
6. Youngs, D. L. Time-dependent multi-material flow with large fluid distortion. *Numerical methods for fluid dynamics* (1982).
7. Francois, M. M. *et al.* A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. en. *Journal of Computational Physics* **213,** 141–173 (Mar. 2006).
8. Renardy, Y. & Renardy, M. PROST: A Parabolic Reconstruction of Surface Tension for the Volume-of-Fluid Method. en. *Journal of Computational Physics* **183,** 400–421 (Dec. 2002).
9. Bornia, G., Cervone, A., Manservisi, S., Scardovelli, R. & Zaleski, S. On the properties and limitations of the height function method in two-dimensional Cartesian geometry. en. *Journal of Computational Physics* **230,** 851–862 (Feb. 2011).
10. Popinet, S. An accurate adaptive solver for surface-tension-driven interfacial flows. en. *Journal of Computational Physics* **228,** 5838–5866 (Sept. 2009).
11. Owkes, M. & Desjardins, O. A mesh-decoupled height function method for computing interface curvature. en. *Journal of Computational Physics* **281,** 285–300 (Jan. 2015).
12. Popinet, S. Numerical Models of Surface Tension. en. *Annual Review of Fluid Mechanics* **50,** 49–75 (Jan. 2018).
13. Meier, M., Yadigaroglu, G. & Smith, B. L. A novel technique for including surface tension in PLIC-VOF methods. en. *European Journal of Mechanics - B/Fluids* **21,** 61–73 (Jan. 2002).

14. Aulisa, E., Manservisi, S. & Scardovelli, R. A mixed markers and volume-of-fluid method for the reconstruction and advection of interfaces in two-phase and free-boundary flows. en. *Journal of Computational Physics* **188,** 611–639 (July 2003).

15. Aulisa, E., Manservisi, S. & Scardovelli, R. A surface marker algorithm coupled to an area-preserving marker redistribution method for three-dimensional interface tracking. en. *Journal of Computational Physics* **197,** 555–584 (July 2004).

Biographical sketch

Yinghe Qi received the B.S. degree in Naval Architecture and Ocean Engineering from Shanghai Jiao Tong University in 2017, and enrolled in the Mechanical Engineering M.S.E program at Johns Hopkins University in the same year. In Dec 2018, he joined Prof. Gretar Tryggvason's research group to work on the computational multiphase flow. In the summer of 2018, he switched to the Ph.D. program at Johns Hopkins and began to focus on experimental multiphase flow under the supervision of Prof. Rui Ni.